VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELGAUM

FILE STRUCTURES LABORATORY WITH MINI PROJECT
(Subject Code: 18ISL67)

MASTER MANUAL

VI-SEMESTER

Prepared By,

Mrs. Divya

Assistant Professor, ISE

AJIET

A J INSTITUTE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

(A unit of Laxmi Memorial Education Trust. (R))

NH - 66, Kottara Chowki, Kodical Cross - 575 006

Vision

To be a centre of excellence in Information Science & Engineering education, research and training to meet the growing needs of the industry and society.

Mission

- To impart theoretical and practical knowledge through the concepts and technologies in Information Science and Engineering
- To foster research, collaboration and higher education with premier institutions and industries.
- Promote innovation and entrepreneurship to fulfill the needs of the society and industry

Course outcomes: The students should be able to:

- Implement operations related to files
- Apply the concepts of file system to produce the given application.
- Evaluate performance of various file systems on given parameters.

Program Specific Outcomes

- Design, implement and maintain the information systems that fulfill the current needs of the industry and society
- Apply computational theory, storage and networking concepts to solve the day to day problems of the world

FILE STRUCTURES LABORATORY WITH MINI PROJECT

[As per Choice Based Credit System (CBCS) scheme]

(Effective from the academic year 2021 -2022)

SEMESTER – VI

Subject Code 18ISL67	IA Marks 20
Number of Lecture Hours/Week 01I + 02P	Exam Marks 80
Total Number of Lecture Hours 40	Exam Hours 03

CREDITS – 02

Course objectives: This course will enable students to

- Apply the concepts of Unix IPC to implement a given function.
- Measure the performance of different file structures
- Write a program to manage operations on given file system.
- Demonstrate hashing and indexing techniques

Description (If any): Design, develop, and implement the following programs

Lab Experiments:

PART A

2. Write a program to read series of names, one per line, from standard input and write these names spelled in reverse order to the standard output using I/O redirection and pipes. Repeat the exercise using an input file specified by the user instead of the standard input and using an output file specified by the user instead of the standard output.

3. Write a program to read and write student objects with fixed-length records and the fields delimited by "|". Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.

4. Write a program to read and write student objects with Variable - Length records using any suitable record structure. Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.

5. Write a program to write student objects with Variable - Length records using any suitable record structure and to read from this file a student record using RRN.

6. Write a program to implement simple index on primary key for a file of student objects. Implement add ( ), search ( ), delete ( ) using the index.

7. Write a program to implement index on secondary key, the name, for a file of student objects. Implement add ( ), search ( ), delete ( ) using the secondary index.

8. Write a program to read two lists of names and then match the names in the two lists using Consequential Match based on a single loop. Output the names common to both the lists.

9. Write a program to read k Lists of names and merge them using k-way merge algorithm with k = 8.

Part B --- Mini project:

Student should develop mini project on the topics mentioned below or similar applications Document processing, transaction management, indexing and hashing, buffer management, configuration management. Not limited to these.

Conduction of Practical Examination:

1. All laboratory experiments from part A are to be included for practical examination.
2. Mini project has to be evaluated for 30 Marks as per 6(b).
3. Report should be prepared in a standard format prescribed for project work.
4. Students are allowed to pick one experiment from the lot.
5. Strictly follow the instructions as printed on the cover page of answer script.
6. Marks distribution:

a) Part A: Procedure + Conduction + Viva:10 + 35 +5 =50 Marks

b) Part B: Demonstration + Report + Viva voce = 15+10+05 = 30 Marks

7. Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

Data Representation in Memory

Record:

A subdivision of a file, containing data related to a single entity.

Field :

A subdivision of a record containing a single attribute of the entity which the record describes.

Stream of bytes:

A file which is regarded as being without structure beyond separation into a sequential set of bytes.

1. Within a program, data is temporarily stored in variables.
2. Individual values can be aggregated into structures, which can be treated as a single variable with parts.
3. In C++, classes are typically used as an aggregate structure.
4. C++ Person class (version 0.1):

```
class Person
{
public:
char  FirstName[11];        char
LastName[11];               char
Address  [21];         char  City
[21];      char State [3];
char ZIP [5];
 };
```

With this class declaration, variables can be declared to be of type Person. The individual fields within a Person can be referred to as the name of the variable and the name of the field, separated by a period (.).

EX : C++ Program:

```
#include<iostream.h>
#include<string.h>
class Person {    public:
char FirstName [11];
char LastName[11];
char Address [31];
char City [21];
```

```
char State [3];
char ZIP [5];
};
void Display (Person);
int main ()
{
 Person Clerk;
 strcpy (Clerk.FirstName, "Fred");
 strcpy (Clerk.LastName, "Flintstone");
strcpy (Clerk.Address, "4444 GranitePlace");
strcpy (Clerk.City, "Rockville");
strcpy (Clerk.State, "MD");
strcpy (Clerk.ZIP, "00001");

Display (Clerk);
}

void Display (Person Someone)
 {
      cout << Someone.FirstName << Someone.LastName<< Someone.Address <<
Someone.City<< Someone.State << Someone.ZIP;
}
```

In memory, each Person will appear as an aggregate, with the individual values being parts of the aggregate

Person					
Clerk					
FirstName	**LastName**	**Address**	**City**	**State**	**ZIP**
Fred	Flintstone	4444 Granite Place	Rockville	MD	0001

The output of this program will be:

FredFlintstone4444 Granite PlaceRockvilleMD00001LilyMunster1313 Mockingbird LaneHollywoodCA90210

Obviously, this output could be improved. It is marginally readable by people, and it would be difficult to program a computer to read and correctly interpret this output.

A Stream File

In the Windows, DOS, UNIX, and LINUX operating systems, files are not internally structured; they are streams of individual bytes.

F	r	E	d		F	l	i	n	t	s	t	o		n	e	4	4	4	4		G	r	a	n	...

The only file structure recognized by these operating systems is the separation of a text file into lines.

- For Windows and DOS, two characters are used between lines, a carriage return (ASCII 13) and a line feed (ASCII 10);
- For UNIX and LINUX, one character is used between lines, a line feed (ASCII 10);

The code in applications programs can, however, impose internal organization on stream files. File processing in C++ is performed using the fstream class. Unlike the FILE structure, fstream is a complete C++ class with constructors, a destructor and overloaded operators.

To perform file processing, you can declare an instance of an fstream object. If you do not yet know the name of the file you want to process, you can use the default constructor.

Unlike the FILE structure, the fstream class provides two distinct classes for file processing. One is used to write to a file and the other is used to read from a file.

Opening a File

In C program, the type FILE is used for a file variable and is defined in the stdio.h file. It is used to define a file pointer for use in file operations. Before we can write to a file, we must open it. What this really means is that we must tell the system that we want to write to a file and what the file name is. We do this with the fopen() function illustrated in the first line of the program. The file pointer, fp in our case, points to the file and two arguments are required in the parentheses, the file name first, followed by the file type. In C++ , two way we can open file by using **constructor** or by using member function **open().**

1. Open a file by constructor:

You can first declare an instance of the **stream** class using one of its constructors from the following syntaxes to open a file :

- **ofstream:** Stream class to write on files
- **ifstream:** Stream class to read from files
- **fstream:** Stream class to both read and write from/to files.

Syntaxes:

ofstream obj(const char* FileName, int FileMode); or
ifstream obj(const char* *FileName*, int *FileMode*); or
fstream obj(const char* *FileName*, int *FileMode*);

These classes are derived directly or indirectly from the classes istream, and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therefore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout, with the only difference that we have to associate these streams with physical files.

The first argument of the constructor, *FileName*, is a constant string that represents the file that you want to open. The *FileMode* argument is a natural number that follows the table of modes as we described below.

Mode	Description
ios::app	If *FileName* is a new file, data is written to it. If *FileName* already exists and contains data, then it is opened, the compiler goes to the end of the file and adds the new data to it.
ios::ate	If *FileName* is a new file, data is written to it and subsequently added to the end of the file. If *FileName* already exists and contains data, then it is opened and data is written in the current position.
ios::in	If *FileName* is a new file, then it gets created fine as an empty file. If *FileName* already exists, then it is opened and its content is made available for processing
ios::out	If *FileName* is a new file, then it gets created fine as an empty file. Once/Since it gets created empty, you can write data to it. If *FileName* already exists, then it is opened, its content is destroyed, and the file becomes as new. Therefore you can create new data to write to it. Then, if you save the file, which is the main purpose of this mode, the new content is saved it.*This operation is typically used when you want to save a file
ios::trunc	If *FileName* already exists, its content is destroyed and the file becomes as new
ios::nocreate	If *FileName* is a new file, the operation fails because it cannot create a new file. If *FileName* already exists, then it is opened and its content is made available for processing
ios::noreplace	If *FileName* is a new file, then it gets created fine. If *FileName* already exists and you try to open it, this operation would fail because it cannot create a file of the same name in the same location.

Let's see an example: // basic file operations

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
        ofstream myfile("example.txt");
         myfile << "Writing this to a file.\n";
        myfile.close();
         return 0;
}
```

OUTPUT:

Writing this to a file.

This code creates a file called example.txt and inserts a sentence into it in the same way we are used to do without, but using the file stream myfile instead.

2. Member function open().

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to *open a file*. An open file is represented within a program by a stream object (an instantiation of one of these classes, in the previous example this was myfile) and any input or output operation performed on this stream object will be applied to the physical file associated to it.

In order to open a file with a stream object we use its member function open():

open (filename, mode);

Where filename is a null-terminated character sequence of type const char * (the same type that string literals have) representing the name of the file to be opened, and mode is an optional parameter with a combination of the flags. flags can be combined using the bitwise operator OR (|). For example, if we want to open the fileexample.bin in binary mode to add data we could do it by the following call to member function open():

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Each one of the open() member functions of the classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

class	default mode parameter
ofstream	ios::out
ifstream	ios::in ios::in | ios::out
fstream	

For ifstream and ofstream classes, ios::in and ios::out are automatically and respectively assumed, even if a mode that does not include them is passed as second argument to the open() member function.

The default value is only applied if the function is called without specifying any value for the mode parameter. If the function is called with any value in that parameter the default mode is overridden, not combined.

File streams opened in binary mode perform input and output operations independently of any format considerations. Non-binary files are known as *text files*, and some translations may occur due to formatting of some special characters (like newline and carriage return characters).

Since the first task that is performed on a file stream object is generally to open a file, these three classes include a constructor that automatically calls the open() member function and has the exact same parameters as this member. Therefore, we could also have declared the previous myfile object and conducted the same opening operation in our previous example by writing:

```
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

Combining object construction and stream opening in a single statement. Both forms to open a file are valid and equivalent.

To check if a file stream was successful opening a file, you can do it by calling to member is_open() with no arguments. This member function returns a bool value of true in the case that indeed the stream object is associated with an open file, or false otherwise:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

Closing a file

When we are finished with our input and output operations on a file we shall close it so that its resources become available again. In order to do that we have to call the stream's member function close(). This member function takes no parameters, and what it does is to flush the associated buffers and close the file:

```
myfile.close();
```

Once this member function is called, the stream object can be used to open another file, and the file is available again to be opened by other processes.

In case that an object is destructed while still associated with an open file, the destructor automatically calls the member function close().

Text files

Text file streams are those where we do not include the ios::binary flag in their opening mode. These files are designed to store text and thus all values that we input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.

Data output operations on text files are performed in the same way we operated with cout:

```
// writing on a text file
#include<iostream>
#include <fstream>
using namespace std;
int main () {
  ofstream myfile ("example.txt");
  if (myfile.is_open())
  {
    myfile << "This is a line.\n";
myfile << "This is another line.\n";
myfile.close();
```

```
[file example.txt] This is
a line.
This is another line.
```

```
 }
 else cout << "Unable to open file";
return 0;
}
```

Data input from a file can also be performed in the same way that we did with cin:

```
// reading a text file
#include <iostream>
#include        <fstream>
#include <string>
using namespace std;

int main () {
string line;
 ifstream myfile ("example.txt");
if (myfile.is_open())
 {
   while ( myfile.good() )
   {
     getline (myfile,line);
     cout << line << endl;
   }
   myfile.close();
 }
 else cout << "Unable to open file";
 return 0;
}
```

```
This is a line.
This is another line.
```

This last example reads a text file and prints out its content on the screen. Notice how we have used a new member function, called good() that returns true in the case that the stream is ready for input/output operations. We have created a while loop that finishes when indeed myfile.good() is no longer true, which will happen either if the end of the file has been reached or if some other error occurred.

Checking state flags

In addition to good(), which checks whether the stream is ready for input/output operations, other member functions exist to check for specific states of a stream (all of them return a bool value):

bad()

Returns true if a reading or writing operation fails. For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left. fail()

Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number. eof()

Returns true if a file open for reading has reached the end.

good()

It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true.

In order to reset the state flags checked by any of these member functions we have just seen we can use the member function clear(), which takes no parameters.

get and put stream pointers

All i/o streams objects have, at least, one internal stream pointer:

ifstream, like istream, has a pointer known as the *get pointer* that points to the element to be read in the next input operation. ofstream, like ostream, has a pointer known as the *put pointer* that points to the location where the next element has to be written.

Finally, fstream, inherits both, the get and the put pointers, from iostream (which is itself derived from bothistream and ostream).

These internal stream pointers that point to the reading or writing locations within a stream can be manipulated using the following member functions:

tellg() and tellp()

These two member functions have no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the get stream pointer (in the case of tellg) or the put stream pointer (in the case of tellp).

seekg() and seekp()

These functions allow us to change the position of the get and put stream pointers. Both functions are overloaded with two different prototypes. The first prototype is:

seekg (position);
seekp (position);

Using this prototype the stream pointer is changed to the absolute position position (counting from the beginning of the file). The type for this parameter is the same as the one returned by functions tellg and tellp: the member type pos_type, which is an integer value.

The other prototype for these functions is:

seekg (offset, direction);
seekp (offset, direction);

Using this prototype, the position of the get or put pointer is set to an offset value relative to some specific point determined by the parameter direction. offset is of the member type off_type, which is also an integer type. And direction is of type seekdir, which is an enumerated type (enum) that determines the point from where offset is counted from, and that can take any of the following values:

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position of the stream pointer
ios::end	offset counted from the end of the stream

The following example uses the member functions we have just seen to obtain the size of a file:

```
// obtaining file size
#include <iostream>
#include <fstream> int
main () {
long begin,end;
  ifstream myfile ("example.txt");
begin = myfile.tellg();    myfile.seekg
(0, ios::end);    end = myfile.tellg();
myfile.close();
  cout << "size is: " << (end-begin) << " bytes.\n";
return 0;
}
```

```
size is: 40 bytes.
```

Buffers and Synchronization

When we operate with file streams, these are associated to an internal buffer of type streambuf. This buffer is a memory block that acts as an intermediary between the stream and the physical file. For example, with an ofstream, each time the member function put (which writes a single character) is called, the character is not written directly to the physical file with which the stream is associated. Instead of that, the character is inserted in that stream's intermediate buffer.

When the buffer is flushed, all the data contained in it is written to the physical medium (if it is an output stream) or simply freed (if it is an input stream). This process is called *synchronization* and takes place under any of the following circumstances:

- **When the file is closed:** before closing a file all buffers that have not yet been flushed are synchronized and all pending data is written or read to the physical medium.
- **When the buffer is full:** Buffers have a certain size. When the buffer is full it is automatically synchronized.
- **Explicitly, with manipulators:** When certain manipulators are used on streams, an explicit synchronization takes place. These manipulators are: flush and endl.
- **Explicitly, with member function sync():** Calling stream's member function sync(), which takes no parameters, causes an immediate synchronization. This function returns an int value equal to -1 if the stream has no associated buffer or in case of failure. Otherwise (if the stream buffer was successfully synchronized) it returns 0.

1. Delineation of Records in a File

Fixed Length Records

A record which is predetermined to be the same length as the other records in the file.

Record 1	Record 2	Record 3	Record 4	Record 5

- The file is divided into records of equal size.
- All records within a file have the same size.
- Different files can have different length records.
- Programs which access the file must know the record length.
- Offset, or position, of the nth record of a file can be calculated.
- There is no external overhead for record separation.
- There may be internal fragmentation (unused space within records.)
- There will be no external fragmentation (unused space outside of records) except for deleted records.
- Individual records can always be updated in place.
- Algorithms for Accessing Fixed Length Records
- Code for Accessing Fixed Length Records Code for Accessing String Records

Delimited Variable Length Records

T E R M S

variable length record

A record which can differ in length from the other records of the file.

delimited record

A variable length record which is terminated by a special character or sequence of characters.

delimiter

A special character or group of characters stored after a field or record, which indicates the end of the preceding unit.

Record 1	#	Record 2	#	Record 3	#	Record 4	#	Record 5	#

- The records within a file are followed by a delimiting byte or series of bytes.
- The delimiter cannot occur within the records.
- Records within a file can have different sizes.
- Different files can have different length records.
- Programs which access the file must know the delimiter.
- Offset, or position, of the nth record of a file cannot be calculated.
- There is external overhead for record separation equal to the size of the delimiter per record.
- There should be no internal fragmentation (unused space within records.)
- There may be no external fragmentation (unused space outside of records) after file updating.
- Individual records cannot always be updated in place.
- Algorithms for Accessing Delimited Variable Length Records
- Code for Accessing Delimited Variable Length Records
- Code for Accessing Variable Length Line Records

Length Prefixed Variable Length Records

110	**Record 1**	**40**	**Record 2**	**100**	**Record 3**	**80**	**Record 4**	**70**	**Record 5**

- The records within a file are prefixed by a length byte or bytes.
- Records within a file can have different sizes.
- Different files can have different length records.
- Programs which access the file must know the size and format of the length prefix.
- Offset, or position, of the nth record of a file cannot be calculated.
- There is external overhead for record separation equal to the size of the length prefix per record.
- There should be no internal fragmentation (unused space within records.)
- There may be no external fragmentation (unused space outside of records) after file updating.
- Individual records cannot always be updated in place.
- Algorithms for Accessing Prefixed Variable Length Records
- Code for Accessing PreFixed Variable Length Records

Delineation of Fields in a Record

- Each record is divided into fields of correspondingly equal size.
- Different fields within a record have different sizes.
- Different records can have different length fields.
- Programs which access the record must know the field lengths.
- There is no external overhead for field separation.
- There may be internal fragmentation (unused space within fields.)

Delimited Variable Length Fields

Field 1	!	Field 2	!	Field 3	!	Field 4	!	Field 5	!

- The fields within a record are followed by a delimiting byte or series of bytes.
- Fields within a record can have different sizes.
- Different records can have different length fields.
- Programs which access the record must know the delimiter.
- The delimiter cannot occur within the data.
- If used with delimited records, the field delimiter must be different from the record delimiter.
- There is external overhead for field separation equal to the size of the delimiter per field.
- There should be no internal fragmentation (unused space within fields.)

Length Prefixed Variable Length Fields

12	Field 1	4	Field 2	10	Field 3	8	Field 4	7	Field 5

- The fields within a record are prefixed by a length byte or bytes.
- Fields within a record can have different sizes.
- Different records can have different length fields.
- Programs which access the record must know the size and format of the length prefix.
- There is external overhead for field separation equal to the size of the length prefix per field.
- There should be no internal fragmentation (unused space within fields.)

Representing Record or Field Length

Record or field length can be represented in either binary or character form.

- The length can be considered as another hidden field within the record.
- This length field can be either fixed length or delimited.
- When character form is used, a space can be used to delimit the length field.
- A two byte fixed length field could be used to hold lengths of 0 to 65535 bytes in binary form.
- A two byte fixed length field could be used to hold lengths of 0 to 99 bytes in decimal character form.
- A variable length field delimited by a space could be used to hold effectively any length.
- In some languages, such as strict Pascal, it is difficult to mix binary values and character values in the same file.
- The C++ language is flexible enough so that the use of either binary or character format is easy.

Tagged Fields

- Tags, in the form "Keyword=Value", can be used in fields.
- Use of tags does not in itself allow separation of fields, which must be done with another method.
- Use of tags adds significant space overhead to the file.
- Use of tags does add flexibility to the file structure.
- Fields can be added without affecting the basic structure of the file.
- Tags can be useful when records have sparse fields - that is, when a significant number of the possible attributes are absent.

Index :

A structure containing a set of entries, each consisting of a key field and a reference field, which is used to locate records in a data file.

Key field :

The part of an index which contains keys.

Reference field:

- The part of an index which contains information to locate records.

- An index imposes order on a file without rearranging the file.
- Indexing works by indirection.

A Simple Index for Entry-Sequenced Files

Simple index

An index in which the entries are a key ordered linear list.

Simple indexing can be useful when the entire index can be held in memory.

Changes (additions and deletions) require both the index and the data file to be changed.

Updates affect the index if the key field is changed, or if the record is moved.

An update which moves a record can be handled as a deletion followed by an addition.

Direct access

Accessing data from a file by record position with the file, without accessing intervening records.

Relative record number

An ordinal number indicating the position of a record within a file.

Primary key

A key which uniquely identifies the records within a file.

Secondary key

A search key other than the primary key.

Secondary index

An index built on a secondary key.

- Secondary indexes can be built on any field of the data file, or on combinations of fields.
- Secondary indexes will typically have multiple locations for a single key.
- Changes to the data may now affect multiple indexes.
- The reference field of a secondary index can be a direct reference to the location of the entry in the data file.
- The reference field of a secondary index can also be an indirect reference to the location of the entry in the data file, through the primary key.
- Indirect secondary key references simplify updating of the file set.
- Indirect secondary key references increase access time.

Cosequential Algorithms

- Initialize (open the input files.)
- Get the first item from each list
- While there is more to do:
  - Compare the current items from each list
  - Based on the comparison, appropriately process one or all items.
  - Get the next item or items from the appropriate list or lists.
  - Based on the whether there were more items, determine if there is more to do.
- Finalize (close the files.)

Match :

The process of forming a list containing all items common to two or more lists.

Cosequential Match Algorithm

- Initialize (open the input and output files.)
- Get the first item from each list.
- While there is more to do:
  - Compare the current items from each list.
  - If the items are equal,
- Process the item.
- Get the next item from each list.
  - Set *more* to true if none of this list is at end of file.
  - If the item from list *A* is less than the item from list *B*,
    - Get the next item from list *A*.

    Set *more* to true if list *A* is not at end-of-file.
  - If the item from list *A* is more than the item from list *B*,
    - Get the next item from list *B*.

    Set *more* to true if list *B* is not at end-of-file. Finalize (close the files.)

Merge

The process of forming a list containing all items in any of two or more lists.

Consequential Merge Algorithm

- Initialize (open the input and output files.)
- Get the first item from each list.
- While there is more to do:
    - Compare the current items from each list.
    - If the items are equal,
        - Process the item.
        - Get the next item from each list.
        - If the item from list *A* is less than the item from list *B*,
        - Process the item from list *A*.
        - Get the next item from list *A*.
    - If the item from list *A* is more than the item from list *B*,
        - Process the item from list *B*.
        - Get the next item from list *B*.
    - Set *more* to false iff all of this lists are at end of file.
- Finalize (close the files.)

1. **Write a program to read series of names, one per line, from standard input and write these names spelled in reverse order to the standard output using I/O redirection and pipes. Repeat the exercise using an input file specified by the user instead of the standard input and using an output file specified by the user instead of the standard output.**

```
#include<fstream>
#include<iostream>
#include<stdlib.h>
#include<string.h>

using namespace std;

char * strrev(char *str1)
{
  int i=0;
  int len=strlen(str1);
 char *str2=new char[20];
          while(len>0)
          {
          str2[i]=str1[--len];
          i++;
          }
  str2[i]='\0';
return str2;
}

main() {
int choice,i,num_name;
fstream fp,fp1;
char name[20][20],filename[20],str[10],filename1[20]; for(;;) {
cout<<"1 : standard input to standard output\n";
cout<<"2 : file to standard output\n";
cout<<"3 : file to file\n";
```

```
cout<<"4 : exit\n";
cout<<"enter your choice:\n";
cin>>choice;
switch(choice)
{
        case 1:cout<<"enter the number of names to read :\n";
                cin>>num_name;
                for(i=1;i<=num_name;i++)
                cin>>name[i];
                  cout<<"\nreversed names are :\n";
                  for(i=1;i<=num_name;i++)
                  cout<<strrev(name[i])<<"\n";
                  break;
        case 2:cout<<"enter the input file name\n";
                cin>>filename;
                fp.open(filename,ios::in);
                if(fp.fail())
                {
                cout<<"cannot open the file\n";
                }
                else
                {
                cout<<"reversed names from the file:\n";
                while(1)
                {
                fp>>str;
                if(fp.fail())
                break;
                cout<<strrev(str)<<endl;
                }
                fp.close();
                }
```

```
                break;
        case 3:cout<<"enter the input file name\n";
                cin>>filename;
                  cout<<"enter the output file name\n";
          cin>>filename1;
                  fp.open(filename,ios::in);
          fp1.open(filename1,ios::out);
                if(fp.fail()||fp1.fail())
                {
                cout<<"cannot open the file\n";
                exit(0);
                }
                while(1)
                {
                fp>>str;
                if(fp.fail())
                break;
                fp1<<strrev(str)<<"\n";
                }
           fp.close();
  fp1.close();
  break;
        default:exit(0);
        }
        }
}
```

Output:

$g++ prog1.cpp

$./a.out

1 : standard input to standard output

2 : file to standard output

3 : file to file 4 : exit enter your choice: 1 enter the number of names to read :
2 beena bincy

reversed names are :
aneeb ycnib
1 : standard input to standard output
2 : file to standard output
3 : file to file 4 : exit enter your choice: 2 enter the input file name 1.txt reversed names from the file:
aytida aneeb ysselb

enter your choice: 3 enter the
input file name 1.txt enter the
output file name
2.txt
1 : standard input to standard output
2 : file to standard output
3 : file to file 4 : exit enter your choice:
4

2. Write a program to read and write student objects with fixed-length records and the fields delimited by "|". Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#define SIZE 50
using namespace std;
fstream file;
class fixedlen_student
{
struct student
```

```
{
char usn[11],name[15],sem[2],dept[5];
};
public: void pack();
void unpack();
void search();
void modify();
};

void fixedlen_student::pack()
{
char buf[SIZE+1];
student s;
cout<<"\nEnter usn,name,sem,dept:";
cin>>s.usn>>s.name>>s.sem>>s.dept;
file.open("student.txt",ios::out|ios::app);
sprintf(buf,"%s|%s|%s|%s|",s.usn,s.name,s.sem,s.dept);
int len=strlen(buf);
while(len<(SIZE))
{
strcat(buf,"-");
len++;
}
strcat(buf,"$");
file<<buf;
file.close();
}

void fixedlen_student::unpack()
{
char buf[SIZE+1];
student s;
fstream file;
```

```
file.open("student.txt",ios::in);
while(!file.eof())
{
file.getline(buf,100,'$');
if(file.eof())
break;
sscanf(buf,"%[^|]|%[^|]|%[^|]|%[^|]",s.usn,s.name,s.sem,s.dept);
cout<<s.usn<<" "<<s.name<<" "<<s.sem<<" "<<s.dept<<endl;
}
file.close();
}

void fixedlen_student::search()
{
char buf[SIZE+1],usn[11];
student s;
int found=0;
file.open("student.txt",ios::in);
cout<<"\nEnter usn to be searched: ";
cin>>usn;
while(!file.eof() && found==0)
{
file.getline(buf,100,'$');
sscanf(buf,"%[^|]|%[^|]|%[^|]|%[^|]",s.usn,s.name,s.sem,s.dept);
if(strcmp(s.usn,usn)==0)
{
found=1;
cout<<"\nRecord found\n";
cout<<s.usn<<" "<<s.name<<" "<<s.sem<<" "<<s.dept<<endl;
break;
}
}
if(found==0)
```

```
cout<<"Record not found\n";
file.close();
}

void fixedlen_student::modify()
{
char buf[SIZE+1],usn[11];
student s;
int n=0,found=0,ch;
file.open("student.txt",ios::in|ios::out);
cout<<"\nEnter usn to be modified:";
cin>>usn;
while(!file.eof())
{
file.getline(buf,100,'$');
sscanf(buf,"%[^|]|%[^|]|%[^|]|%[^|]",s.usn,s.name,s.sem,s.dept);
if(strcmp(s.usn,usn)==0)
{
found=1;
cout<<"\nKeyfound\n1:Modifyname\n2:modifysem\n3:MOdifyDept\nEnter your choice:";
cin>>ch;
switch(ch)
{
case 1:cout<<"\nENter name:";cin>>s.name; break;
case 2:cout<<"\nENter sem:";cin>>s.sem; break;
case 3:cout<<"\nEnter dept:";cin>>s.dept; break;
default:break;
}
sprintf(buf,"%s|%s|%s|%s|",s.usn,s.name,s.sem,s.dept);
int len=strlen(buf);
while(len<(SIZE))
{
strcat(buf,"-");
```

```
len++;
}
strcat(buf,"$");
file.seekp((n*(SIZE+1)),ios::beg);
file<<buf;
break;
}
n++;
}
if(found==0)
cout<<"Record not found\n";
file.close();
}

main()
{
fixedlen_student R;
int ch;
for(;;)
{
cout<<"\n1:pack\t2:unpack\t3:search\t4:modify\t5:Exit\nEnteryourchoice:";
cin>>ch;
switch(ch)
{
case 1: R.pack();
break;
case 2: R.unpack();
break;
case 3: R.search();
break;
case 4: R.modify();
break;
default:exit(0);
```

```
}
}
}
```

Output:

```
$ g++ prog2.cpp
$ ./a.out
1:pack 2:unpack 3:search 4:Modify 5:Exit Enter your choice:1
Enter usn,name,sem,dept: 001
ajay
6
is
1:pack 2:unpack 3:search 4:Modify 5:Exit Enter your choice:1
Enter usn,name,sem,dept:018
thanmayee
6
is

1:pack 2:unpack 3:search 4:Modify 5:Exit Enter your choice:2
001 ajay 6 is
018 thanmayee 6 is

1:pack 2:unpack 3:search 4:Modify 5:Exit Enter your choice:3
Enter usn to be searched: 001
Record not found

1:pack 2:unpack 3:search 4:Modify 5:Exit Enter your choice:3
Enter usn to be searched: 018
Record found
018 thanmayee 6 is

1:pack 2:unpack 3:search 4:Modify 5:Exit Enter your choice:4
Enter usn to be modified: 001
```

```
Key found
1:Modifyname
2:modifysem
3:MOdifyDept
Enter your choice:1
ENter name:ashwini

1:pack 2:unpack 3:search 4:Modify 5:Exit Enter your choice:2
001 ashwini 6 is
018 thanmayee 6 is

1:pack 2:unpack 3:search 4:Modify 5:Exit Enter your choice:5
```

3. Write a program to read and write student objects with Variable - Length records using any suitable record structure. Implement pack ( ), unpack ( ), modify ( ) and search ( ) methods.

```
#include<iostream>
#include<string.h>
#include<fstream>
#include<stdio.h>
#include<stdlib.h>
#define RECORDSIZE 50
using namespace std;
fstream file;
class varlen_student
{
  struct student
  {
        char usn[11],name[15],branch[5],college[15];
  };
  public:
        void pack();
        void unpack();
```

```
        void search();
        void modify();
};
void varlen_student::pack()
{
  student s;
  char buff[RECORDSIZE+1];
  cout<<"enter usn,name,branch and college\n";
  cin>>s.usn>>s.name>>s.branch>>s.college;
  sprintf(buff,"%s|%s|%s|%s|$",s.usn,s.name,s.branch,s.college);
  file.open("pg.txt",ios::out|ios::app);
  file<<buff;
  file.close();
}

void varlen_student::unpack()
{
  student s;
  char buff[RECORDSIZE+1];
  file.open("pg.txt",ios::in);
  while(!file.eof())
  {
        file.getline(buff,RECORDSIZE,'$');
        if(file.eof())
              break;
        sscanf(buff,"%[^|]|%[^|]| %[^|]|%[^|]|$", s.usn,s.name,s.branch,s.college);
        cout<<"\n----------------\nusn\t:"<<s.usn<<"\nname\t:"<<s.name<<"\nbranch\t:"
        <<s.branch<<"\ncollege:"<<s.college<<endl;
  }
  file.close();
}

void varlen_student::search()
```

```
{
  student s;
  int found=0;
  char buff[RECORDSIZE+1];
  char usn[11];
  file.open("pg.txt",ios::in);
  cout<<"\nEnter the usn to be searched:";
  cin>>usn;
  while(!file.eof() && found==0)
  {
          file.getline(buff,RECORDSIZE,'$');
          if(file.eof())
                  break;
          sscanf(buff,"%[^|]|%[^|]|%[^|]|%[^|]|$",s.usn,s.name,s.branch,s.college);
          if(strcmp(s.usn,usn)==0)
          {
                  found=1;
                  cout<<"\nRecord found";

                  cout<<"\n----------------
\nusn\t:"<<s.usn<<"\nname\t:"<<s.name<<"\nbranch\t:"<<s.branch<<"\ncollege:"<<s.college<<endl;
                  break;
          }
  }
  if(found==0)
          cout<<"\nRecord not found\n";
  file.close();
}

void varlen_student::modify()
{
  int n=0,found=0;
  student s;
```

```
  char usn[11];
  char buff[RECORDSIZE+1];
  fstream newfile;
  file.open("pg.txt",ios::in);
  newfile.open("temp.txt",ios::out|ios::app);
  cout<<"\nEnter the usn to be modified:";
  cin>>usn;
  while(!file.eof())
  {
        file.getline(buff,RECORDSIZE,'$');
        if(file.eof())
              break;
        sscanf(buff,"%[^|]|%[^|]|%[^|]|%[^|]|$",s.usn,s.name,s.branch,s.college);
        if(strcmp(s.usn,usn)==0)
        {
              found=1;
              cout<<"\nRe-Enter student details: ";
              cout<<"enter usn,name,branch and college\n";
              cin>>s.usn>>s.name>>s.branch>>s.college;
        }
        sprintf(buff,"%s|%s|%s|%s|$",s.usn,s.name,s.branch,s.college);
        newfile<<buff;
  }
  if(!found)
        cout<<"\n\nRecord to be modified does not exist in file\n";
  file.close();
  newfile.close();
  remove("pg.txt");
  rename("temp.txt","pg.txt");
}
```

```
int main()
{
  int ch;
  varlen_student b;
  for(;;)
  {
        cout<<"\n1.pack\t2.unpack\t3.search\t4.modify\t5.exit\nEnter your choice:";
        cin>>ch;
        switch(ch)
        {
              case 1: b.pack();
                    break;
              case 2: b.unpack();
                    break;
              case 3: b.search();
                    break;
              case 4: b.modify();
                    break;
              default:exit(0);
        }
  }
}
```

Output:

```
$ g++ prog3.cpp
./a.out
1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:1
enter usn,name,branch and college
001
harsha
is
AJIET
```

1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:1
enter usn,name,branch and college
010
chethan
is
AJIET

1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:1
enter usn,name,branch and college
017
suhan
is
AJIET

1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:2
---
usn : 001
name :harsha
branch :is
college: AJIET
---
usn :010
name :chethan
branch :is
college: AJIET
---
usn :017
name :suhan
branch :is
college: AJIET
1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:3
Enter the usn to be searched: 001
Record found

---

usn :001

name :harsha

branch :is

college: AJIET

1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:3

Enter the usn to be searched:000

Record not found

1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:4

Enter the usn to be modified:017

Re-Enter student details: enter usn,name,branch and college

017

akshatha

is

AJIET

1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:2

---

usn :001

name :harsha

branch :is

college: AJIET

---

usn :010

name :chethan

branch :is

college: AJIET

---

usn :017

name :akshatha

branch :is

college: AJIET

1.pack 2.unpack 3.search 4.modify 5.exit Enter your choice:5

4. Write a program to write student objects with Variable - Length records using any suitable record structure and to read from this file a student record using RRN.

```
#include<stdio.h>
#include<stdlib.h>
#include<fstream>
#include<iostream>
#include<string.h>
using namespace std;
fstream file;
class varlen_student
{
  struct student
  {
        char usn[11],name[15],sem[2],dept[5];
  };
  public: void pack();
        void unpack();
        void search();
};
void varlen_student::pack()
{
  int count=0;
  char b[100],c[150],temp[100];
  student s;
  cout<<"enter usn, name, sem, dept";
  cin>>s.usn>>s.name>>s.sem>>s.dept;
  file.open("stu.txt",ios::in|ios::app);
  while(!file.eof())
  {
```

```
            file.getline(temp,100,'#');
            if(file.eof())
                  break;
            count++;
  }
            file.close();
  file.open("stu.txt",ios::out|ios::app);
  sprintf(b,"%s|%s|%s|%s|#",s.usn,s.name,s.sem,s.dept);
  sprintf(c,"|%d|%d$",count,strlen(b));
  strcat(c,b);
  file<<c;
  file.close();
}
void varlen_student::unpack()
{
  char b[100];
  student s;
  file.open("stu.txt",ios::in);
  while(!file.eof())
  {
            file.getline(b,100,'$');
            file.getline(b,100,'#');
            if(file.eof())
                  break;
            sscanf(b,"%[^|]|%[^|]|%[^|]|%[^|]|#",s.usn,s.name,s.sem,s.dept);
            cout<<s.usn<<" "<<s.name<<" "<<s.sem<<" "<<s.dept<<endl;
  }
  file.close();
}

void varlen_student::search()
{
  char b[100],rrn[11],temp[100],len[10];
```

```
  int flag=0;
  student s;
  file.open("stu.txt",ios::in);
  cout<<"\n enter a rrn to be searched\n";
  cin>>rrn;
  while(!file.eof())
  {
        file.getline(b,100,'#');
        if(file.eof())
              break;
        sscanf(b,"|%[^|]",temp);
        if(strcmp(temp,rrn)==0)
        {
              flag=1;
              //|100|13$1|diya|3|is|#

  sscanf(b,"|%[^|]|%[^$]$%[^|]|%[^|]|%[^|]|%[^|]|#",temp,len,s.usn,s.name,s.sem,s.dept);
              cout<<s.usn<<" "<<s.name<<" "<<s.sem<<" "<<s.dept<<"\n";
              break;
        }
  }
  if(flag==0)
        cout<<"\n key not found\n";
  file.close();
}

main()
{
  varlen_student b;
  int ch;
  for(;;)
  {
        cout<<"\n1:pack\t2:unpack\t3:search\t4:exit\nenter your choice:";
```

```
            cin>>ch;
            switch(ch)
            {
                  case 1: b.pack();
                        break;
                  case 2: b.unpack();
                        break;
                  case 3: b.search();
                        break;
                  default: exit(0);
            }
      }
}
```

Output:

```
$ g++ prog4.cpp
./a.out
1:pack 2:unpack 3:search 4:exit enter your choice:1
enter usn, name, sem, dept4jk14is007
chethan
6
is

1:pack 2:unpack 3:search 4:exit enter your choice:1
enter usn, name, sem, dept4jk14is017
harsha
6
is

1:pack 2:unpack 3:search 4:exit enter your choice:2
4jk14is007 chethan 6 is
4jk14is017 harsha 6 is
1:pack 2:unpack 3:search 4:exit enter your choice:3
```

```
enter a rrn to be searched
0
4jk14is007 chethan 6 is

1:pack 2:unpack 3:search 4:exit enter your choice:3
enter a rrn to be searched
1
4jk14is017 harsha 6 is

1:pack 2:unpack 3:search 4:exit enter your choice:3
enter a rrn to be searched
121
key not found

1:pack 2:unpack 3:search 4:exit enter your choice:4
```

5. Write a program to implement simple index on primary key for a file of student objects. Implement add ( ), search ( ), delete ( ) using the index.

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<fstream>
using namespace std;
fstream data,indx;
char pri[125][15];
int ind[125],count=0;
int flag_updt=0;

class Index_student
{
```

```
struct student
{
char regno[11];
char name[20];
char address[30];
char sem[2];
char branch[5];
char college[10];
};
public:
void Insert();
void Delete();
void Display();
void LoadIndex();
void WriteIndex();
};

void Index_student::Insert()
{
char buf[100];
int pos;
student s;
data.open("std.txt",ios::out|ios::app);
data.seekg(0,ios::end);
pos=data.tellg();
cout<<"--------------------------\n";
cout<<"ENTER RECORD DETAILS\n";
cout<<"--------------------------\n";
cout<<"RegNo : ";
cin>>s.regno;
for(int i=0;i<count;i++)
{
if(strcmp(pri[i],s.regno) == 0)
```

```
{
cout<<endl;
cout<<"DUPLICATE RECORD !!!";
data.close();
return;
}
}
cout<<"Name : ";
cin>>s.name;
cout<<"Address : ";
cin>>s.address;
cout<<"Sem : ";
cin>>s.sem;
cout<<"Branch : ";
cin>>s.branch;
cout<<"College : ";
cin>>s.college;
sprintf(buf,"|%s|%s|%s|%s|%s|%s #",s.regno,s.name,s.address,s.sem,s.branch,s.college);
data<<buf;
strcpy(pri[count],s.regno);
ind[count]=pos;
count++;
flag_updt=1;
data.close();
}

void Index_student::Delete()
{
char reg[20],buf[100],usn[15];
int pos;
int flag=0;
cout<<"ENTER URN:";
cin>>reg;
```

```
data.open("std.txt",ios::in|ios::out);
for(int i=0;i<count;i++)
if(strcmp(pri[i],reg) == 0)
{
flag=1;
data.seekg(ind[i],ios::beg);
data.getline(buf,100,'#');
buf[0]='*';
pri[i][0]='*';
data.seekg(ind[i],ios::beg);
data<<buf;
cout<<"RECORD DELETED\n\n";
flag_updt=1;
break;
}
if(flag==0)
{
cout<<"RECORD NOT FOUND !!!\n\n";
}
data.close();
}

void Index_student::Display()
{
int flag=0;
char buf[100];
char regno[20];
student s;
cout<<"ENTER URN: ";
cin>>regno;
data.open("std.txt",ios::in|ios::out);
for(int i=0;i<count;i++)
{
```

```
if(strcmp(pri[i],regno) == 0)
{
flag=1;
data.seekg(ind[i]);
data.getline(buf,100,'#');
sscanf(buf,"|%[^|]|%[^|]|%[^|]|%[^|]|% [^|]#",s.regno,s.name,s.address,s.sem,s.branch,s.college);
cout<<"-------------------------";
cout<<endl<<"RECORD DETAILS";
cout<<endl<<"-------------------------";
cout<<endl<<"URN : "<<s.regno;
cout<<endl<<"NAME : "<<s.name;
cout<<endl<<"ADDRESS : "<<s.address;
cout<<endl<<"SEMESTER : "<<s.sem;
cout<<endl<<"BRANCH : "<<s.branch;
cout<<endl<<"COLLEGE : "<<s.college;
cout<<"\n";
}
}
if(flag==0)
cout<<"RECORD DOES NOT EXISTS\n\n";
data.close();
}

void Index_student::LoadIndex()
{
char buf[100],temp[100];
indx.open("indx.txt",ios::in);
while(indx)
{
indx.getline(buf,100,'#');
if(indx.eof())
break;
sscanf(buf,"|%[^|]|%[^|]",pri[count],temp);
```

```
ind[count]=atoi(temp);
count++;
}
indx.close();
}

void Index_student::WriteIndex()
{
char buf[100];
indx.open("indx.txt",ios::out);
for(int i=0;i<count;i++)
{
sprintf(buf,"%d",ind[i]);
indx<<"|"<<pri[i]<<"|"<<buf<<"|"<<"#";
}
indx.close();
}

main()
{
Index_student I;
I.LoadIndex();
int ch;
while(1)
{
cout<<"1->INSERT RECORD\n";
cout<<"2->DELETE RECORD\n";
cout<<"3->DISPLAY INDIVIDUAL RECORD\n";
cout<<"4->QUIT\n";
cout<<"ENTER YOUR CHOICE:";
cin>>ch;
switch(ch)
{
```

```
case 1:I.Insert();
break;
case 2:I.Delete();
break;
case 3:I.Display();
break;
case 4:if(flag_updt==1)
I.WriteIndex();
exit(0);
}
}

}
```

Output:

```
$ g++ prog5.cpp
$ ./a.out
1->INSERT RECORD 2->DELETE RECORD 3->DISPLAY INDIVIDUAL RECORD 4->QUIT ENTER
YOUR CHOICE:1
---------------------------
ENTER RECORD DETAILS
---------------------------
RegNo : 4JK14is030
Name : vaishnavi
Address : MANGALORE
Sem : 6
Branch : is
College : AJIET
1->INSERT RECORD 2->DELETE RECORD 3->DISPLAY INDIVIDUAL RECORD 4->QUIT ENTER
YOUR CHOICE:1
---------------------------
```

ENTER RECORD DETAILS

RegNo : 4JK14is025

Name : sampreethi

Address : mangalore

Sem : 6

Branch : is

College : AJIET

1->INSERT RECORD 2->DELETE RECORD 3->DISPLAY INDIVIDUAL RECORD 4->QUIT ENTER YOUR CHOICE:2

ENTER URN:4JK14is025

RECORD DELETED

1->INSERT RECORD 2->DELETE RECORD 3->DISPLAY INDIVIDUAL RECORD 4->QUIT ENTER YOUR CHOICE:1

ENTER RECORD DETAILS

RegNo : 4JK14is023

Name : raksha

Address : MANGALORE

Sem : 5

Branch : is

College : AJIET

1->INSERT RECORD 2->DELETE RECORD 3->DISPLAY INDIVIDUAL RECORD 4->QUIT ENTER YOUR CHOICE:3

ENTER URN: 4JK14is025

RECORD DOES NOT EXISTS

1->INSERT RECORD 2->DELETE RECORD 3->DISPLAY INDIVIDUAL RECORD 4->QUIT ENTER YOUR CHOICE:3

ENTER URN: 4JK14is023

RECORD DETAILS

URN : 4JK14is023
NAME : raksha
ADDRESS : MANGALORE
SEMESTER : 5
BRANCH : is
COLLEGE : AJIET

1->INSERT RECORD 2->DELETE RECORD 3->DISPLAY INDIVIDUAL RECORD 4->QUIT ENTER YOUR CHOICE:4

6. Write a program to implement index on secondary key, the name, for a file of student objects. Implement add ( ), search ( ), delete ( ) using the secondary index.

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<fstream>
using namespace std;
fstream data,pri_ind,sec_ind;


char pri[125][15],sec[125][40],usn[125][20];
int ind[125],count;

class Sec_Index_student
{
  struct student
  {
        char regno[11],name[20],address[30],sem[2],branch[5],college[10];
  };
```

```
  public:
        void Insert();
        void Delete();
        void Display();
        void LoadIndex();
        void WriteIndex();
};

void Sec_Index_student::Insert()
{
  char buf[100];
  int pos,mid;
  student s;
  data.open("stu.txt",ios::out|ios::app);
  data.seekg(0,ios::end);
  pos=data.tellg();
  cout<<"ENTER RECORD DETAILS\n";
  cout<<"RegNo     : ";
  cin>>s.regno;
  for(int i=0;i<count;i++)
  {
  if(strcmp(s.regno,pri[i])==0)
        {
              cout<<"DUPLICATE RECORD!!!";
              data.close();
              return;
        }
  }
  cout<<"Name:";
  cin>>s.name;
  cout<<"Address:";
  cin>>s.address;
  cout<<"Sem:";
```

```
  cin>>s.sem;
  cout<<"Branch:";
  cin>>s.branch;
  cout<<"College:";
  cin>>s.college;
  sprintf(buf,"|%s|%s|%s|%s|%s|%s|#",s.regno,s.name,s.address,s.sem,s.branch,s.college);
  data<<buf;
  strcpy(pri[count],s.regno);//pri record
  ind[count]=pos;
  strcpy(sec[count],s.name);//sec record
  strcpy(usn[count],s.regno);
  count++;
  this->WriteIndex();
  data.close();
}

void Sec_Index_student::Delete()
{
  char buf[100],name[20];
  int flag=0,indx=count;
  cout<<"enter the name";
  cin>>name;
  cout<<count<<endl;
  data.open("stu.txt",ios::in|ios::out);
  for(int i=0;i<count;i++)
          {
                  if(strcmp(sec[i],name)==0)
          {
          indx=i;
                          break;
          }
  }
  if(indx<count) {
```

```
for(int i=0;i<count;i++)
{
        if(strcmp(usn[indx],pri[i])==0)
                {
                        flag=1;
                        data.seekg(ind[i],ios::beg);
                        data.getline(buf,100,'#');
                        buf[0]='*';//delete stu rec
                        sec[indx][0]='*';
                        pri[i][0]='*';//delete pri index
                        data.seekg(ind[i],ios::beg);
                        data<<buf;//write deleted record
                        cout<<endl<<"RECORD DELETED\n";
                        this->WriteIndex();
                        data.close();
                return;
                }
  }
}


  if(flag==0)
  {
        cout<<"\n RECORD NOT FOUND"<<endl;
  }
  data.close();
}

void Sec_Index_student::Display()
{
  char buf[100],name[20];
  int flag=0,indx=count;
  student s;
```

```
  cout<<"enter the name";
  cin>>name;
  data.open("stu.txt",ios::in|ios::out);
  for(int i=0;i<count;i++)     {        if(strcmp(sec[i],name)==0) { indx=i;    break; } }
  for(int i=0;i<count;i++)
          {
                  if(strcmp(usn[indx],pri[i])==0)
                  {
                          flag=1;
                          data.seekg(ind[i],ios::beg);
                          data.getline(buf,100,'#');

  sscanf(buf,"|%[^|]|%[^|]|%[^|]|%[^|]|%[^|]#",s.regno,s.name,s.address,s.sem,s.branch,s.college);
                          cout<<"\nRECORD DETAILS"<<endl;
                          cout<<endl<<"URN          : "<<s.regno;
                          cout<<endl<<"NAME         : "<<s.name;
                          cout<<endl<<"ADDRESS      : "<<s.address;
                          cout<<endl<<"SEMESTER     : "<<s.sem;
                          cout<<endl<<"BRANCH       : "<<s.branch<<endl;
                          data.close();
                          return;
                          }
          }


  if(flag==0)
          cout<<"RECORD DOES NOT EXIST\n";
  data.close();
}

void Sec_Index_student::LoadIndex()
{
  char buf[100],temp[100];
```

```
  count=0;
  pri_ind.open("pri_index.txt", ios::in);
  sec_ind.open("sec_index.txt",ios::in);
  while(pri_ind)
  {
        pri_ind.getline(buf,100,'#');
        if(pri_ind.eof())
              break;
        sscanf(buf,"|%[^|]|%[^|]",pri[count],temp);
        ind[count]=atoi(temp);
        count++;
  }
  count=0;
  while(sec_ind)
  {
        sec_ind.getline(buf,100,'#');
        if(sec_ind.eof())
              break;
        sscanf(buf,"|%[^|]|%[^|]",sec[count],usn[count]);
        count++;
  }
  pri_ind.close();
  sec_ind.close();
}

void Sec_Index_student::WriteIndex()
{
  char buf[100];
  pri_ind.open("pri_index.txt", ios::out);
  sec_ind.open("sec_index.txt",ios::out);
  //bubble_sort();
  for(int i=0;i<count;i++)
  {
```

```
            pri_ind<<"|"<<pri[i]<<"|"<<ind[i]<<"|"<<"#";//pri record writing
  }
  for(int i=0;i<count;i++)
  {
          sec_ind<<"|"<<sec[i]<<"|"<<usn[i]<<"|"<<"#";//sec record writing
  }
  pri_ind.close();
  sec_ind.close();
}

main()
{
  Sec_Index_student I;
  char name[20],regno[20];
  I.LoadIndex();
  int ch;
  while(true)
  {
          int c;
          cout<<"   1->INSERT RECORD\n";
          cout<<"   2->DELETE RECORD\n";
          cout<<"   3->DISPLAY INDIVIDUAL RECORD\n";
          cout<<"   4->exit\n";
          cout<<"ENTER YOUR CHOICE :";
          cin>>c;
          switch(c)
          {
                  case 1:I.Insert();
                      break;
                  case 2:I.Delete();
                      break;
                  case 3:I.Display();
                      break;
```

```
                default:  exit(0);
            }
    }
}
```

Output:

```
$ g++ prog6.cpp
./a.out

  1->INSERT RECORD
  2->DELETE RECORD
  3->DISPLAY INDIVIDUAL RECORD
  4->exit
ENTER YOUR CHOICE :1
ENTER RECORD DETAILS
RegNo     : 99
Name:divya
Address:Mangalore
Sem:1
Branch:CS
College:AJ
  1->INSERT RECORD
  2->DELETE RECORD
  3->DISPLAY INDIVIDUAL RECORD
  4->exit
ENTER YOUR CHOICE :1
ENTER RECORD DETAILS
RegNo     : 34
Name:geeta
Address: Mangalore
Sem:4
Branch:EC
```

```
College:AJ
  1->INSERT RECORD
  2->DELETE RECORD
  3->DISPLAY INDIVIDUAL RECORD
  4->exit
ENTER YOUR CHOICE :1
ENTER RECORD DETAILS
RegNo     : 89
Name:seema
Address: Mangalore
Sem:1
Branch:CS
College:AJ
  1->INSERT RECORD
  2->DELETE RECORD
  3->DISPLAY INDIVIDUAL RECORD
  4->exit
ENTER YOUR CHOICE :2
enter the name: seema

RECORD DELETED
  1->INSERT RECORD
  2->DELETE RECORD
  3->DISPLAY INDIVIDUAL RECORD
  4->exit
ENTER YOUR CHOICE :3
enter the name: seema
RECORD DOES NOT EXIST
  1->INSERT RECORD
  2->DELETE RECORD
  3->DISPLAY INDIVIDUAL RECORD
  4->exit
ENTER YOUR CHOICE :3
```

```
enter the name geeta

RECORD DETAILS

URN         : 34
NAME        : geeta
ADDRESS     : Mangalore
SEMESTER    : 4
BRANCH      : CS
  1->INSERT RECORD
  2->DELETE RECORD
  3->DISPLAY INDIVIDUAL RECORD
  4->exit
ENTER YOUR CHOICE :4
```

7. Write a program to read two lists of names and then match the names in the two lists using Consequential Match based on a single loop. Output the names common to both the lists.

```
#include<iostream>
#include<string.h>
#include<fstream>
#include<stdio.h>
using namespace std;
fstream list0, list1, list2;
class intersect
{
  public:
        void sort(char [25][30],int);
        void input(char [25][30],int);
};

void intersect::sort(char s[25][30],int count)
{
```

```
  int i,j;
  char temp[40];
  for(i=1;i<count;i++)
        for(j=0;j<count-i;j++)
              if(strcmp(s[j],s[j+1])>0)
              {
                    strcpy(temp,s[j]);
                    strcpy(s[j],s[j+1]);
                    strcpy(s[j+1],temp);
              }
}

void intersect::input(char str[25][30],int lnum)
{
  int i,j;
  if(lnum==0)
        list0.open("list0.txt",ios::out);
  else
        list1.open("list1.txt",ios::out);
  for(i=0;;i++)
  {
        cin>>str[i];
        if(strcmp(str[i],"#")==0)
              break;
  }
  sort(str,i);
  if(lnum==0)
  {
        for(j=0;j<i;j++)
        {
        list0<<str[j]<<"\n";
        }
  }
```

```
  else
  {
        for(j=0;j<i;j++)
        {
        list1<<str[j]<<"\n";
        }
  }

  list0.close();
  list1.close();
}

main()
{
  char str0[25][30],str1[25][30];
  char buf1[30],buf2[30];
  intersect I;
  cout<<"Enter the names of list1(To terminate enter string #)\n";
  I.input(str0,0);
  cout<<"Enter the names of list2(To terminate enter string #)\n";
  I.input(str1,1);
  list0.open("list0.txt",ios::in);
  list1.open("list1.txt",ios::in);
  list2.open("list2.txt",ios::out);
  list0.getline(buf1,50,'\n');
  list1.getline(buf2,50,'\n');
  while(!list0.eof() && !list1.eof())
  {
        if(strcmp(buf1,buf2)==0)
        {
              list2<<buf1<<endl;
              list0.getline(buf1,50,'\n');
              list1.getline(buf2,50,'\n');
```

```
            }
            else if(strcmp(buf1,buf2)<0)
                        list0.getline(buf1,50,'\n');
            else
                        list1.getline(buf2,50,'\n');
  }
  if(strcmp(buf1,buf2)==0)
            cout<<buf1<<endl;
  list0.close();
  list1.close();
  list2.close();
  list2.open("list2.txt",ios::in);
  list2.getline(buf1,50,'\n');
  if(strcmp(buf1,"\0")==0)
            cout<<" No matching strings found:"<<endl;
  else
  {
            cout<<"Names in both the lists are"<<endl;
            while(1)
            {
                        cout<<buf1<<endl;
                        list2.getline(buf1,50,'\n');
                        if(list2.eof())                         break;
            }
  }
}
```

Output:

Enter the names of list1(To terminate enter string #)

harsha

deeksha

akshatha

```
manasa
pooja
#
Enter the names of list2(To terminate enter string #)
mithun
yashas
shashank
manasa
deeksha
#
Names in both the lists are
deeksha
manasa
```

8. Write a program to read k Lists of names and merge them using k-way merge algorithm with k = 8.

```
#include<stdio.h>
#include<string.h>
#include<fstream>
#include<iostream>
using namespace std;

int main()
{
  char name[20][20],usn[20][20], temp[10], temp1[20];
  fstream list[8];
  fstream file;
  int flag=0, n,i,j;
  char ip_fname[8][8]={"1.txt","2.txt","3.txt","4.txt","5.txt","6.txt","7.txt","8.txt"};
  cout<<"Enter the no of records: ";
        cin>>n;
  for(i=0;i<8;i++)
```

```
list[i].open(ip_fname[i],ios::out);
for(i=0;i<n;i++)
{
        cout<<"Name:";
        cin>>name[i];
        cout<<"Usn:";
        cin>>usn[i];
        list[i%8]<<name[i]<<"|"<<usn[i]<<"\n";
}
for(i=0;i<8;i++)
        list[i].close();
for(i=0;i<n;i++)
{
        for(j=i+1;j<n;j++)
        {
                if(strcmp(name[i],name[j])>0)
                {
                        strcpy(temp,name[i]);
                        strcpy(name[i],name[j]);
                        strcpy(name[j], temp);
                        strcpy(temp1,usn[i]);
                        strcpy(usn[i],usn[j]);
                        strcpy(usn[j], temp1);
                }
        }

}

file.open("merge.txt",ios::out);
cout<<"sorted name:";
for(i=0;i<n;i++)
{
        if(strcmp(name[i],name[i-1])==0) continue;
```

```
            file<<name[i]<<"|"<<usn[i]<<"\n";
            cout<<name[i]<<"|"<<usn[i]<<"\n";
            flag=1;
  }
  if(flag=1)
  cout<<"completed";
  else
  cout<<"not completed";
}
```

output:

```
$ gedit prog8.cpp
$ g++ prog8.cpp
$ ./a.out

 Enter the no of records: 10
Name:divya
Usn:100
Name:reena
Usn:101
Name:seema
Usn:102
Name:raj
Usn:103
Name:usha
Usn:104
Name:tina
Usn:105
Name:gowri
Usn:106
Name:anusha
Usn:107
Name:shilpa
```

Usn:108

Name:preethi

Usn:109

sorted name:

anusha|107

divya|100

gowri|106

preethi|109

raj|103

reena|101

seema|102

shilpa|108

tina|105

usha|104

completed

VIVA QUESTIONS

1. What is File Structure?
2. What is a File?
3. What is a field?
4. What is a Record?
5. What is fixed length record?
6. What is RRN?
7. What is Variable length record?
8. What are the different modes of opening a file?
9. What is ifstream()?
10. What is ofstream()?
11. What is the difference between read() and getline()?
12. How to close a file? What happens if a file is not closed?
13. What is Hashing? What is its use?
14. Explain any one collision resolution technique.
15. What is Btree? What is B+tree?

16. Differentiate between Logical and Physical file
17. What is the use of seekg() and seekp()?
18. Explain the different way of write data to a file.
19. Explain the different way of write data to a file.
20. What is meant by Indexing?
21. What is multilevel indexing?
22. What is File descriptor?
23. What is Fragmentation? What is internal fragmentation?
24. What is DMA?
25. What is a delimeter?
26. Define direct access.
27. Define sequential access.
28. What is the need of packing the record before writing into the file?
29. Explain ios::trunk and ios::nocreate
30. What is the use of End-of-file (EOF)?
31. What are stdin, stdout and stderr?